Security Best Practices

Introduction

In the Play+ ecosystem, security is not an afterthought; it's a foundational layer woven into every component and interaction. This helper is based on the concept of Security by Design , enabling teams to proactively mitigate common web vulnerabilities without needing to be security experts.

The playguard helper provides a simple, unified API for complex security tasks like input sanitization, secure token management, and Cross-Site Scripting (XSS) prevention. This directly supports our core design pillars by making applications more Intuitive for developers to secure, more Adaptive and resilient against threats, and more Inclusive by protecting the data and privacy of all users.

Package Info

The Play+ security helper is provided via the @playplus/security package and is a core component of the Golden Path starter kit.

Description

Package / Path	Golden Path (Recommended)
Pre-installed	/system/play.security.ts
Uplift Path	npm install @playplus/security

Folder Reference

The security helper and its configuration follow our standardized folder structure for core system logic.

File / Directory	Purpose & Guidelines
system/play.security.ts	The core security service. It provides methods for sanitization and authentication lifecycle management.

File / Directory	Purpose & Guidelines
system/api/apiProxy.ts	The secure network gateway that uses playguard internally. Direct fetch() is not permitted.
config/play.security.config.json	User-overridable configuration for Content Security Policy (CSP), CSRF tokens, and auth storage strategy.

Helper - Pillars Alignment

The playguard helper is a direct implementation of our pillars, focused on creating safe and trustworthy digital experiences.

Pillar	How This Helper Aligns
Intuitive	Primary Pillar: Abstracts complex security patterns (like token refresh, input sanitization) into simple, memorable methods.
Adaptive	Builds a resilient application that can gracefully handle authentication state changes and defend against common attacks.
Inclusive	Protects the data and privacy of all users by default, fostering a safe and trustworthy environment for everyone.

Helper Overview

The playguard helper is your application's embedded security engine. Its purpose is to abstract the plumbing of web security, making it easy for developers to build safe applications by default. It provides a simple API that handles complex operations in the background.

It automates and simplifies:

• Input Sanitization : Prevents XSS attacks by cleaning user-provided strings and HTML.
• Authentication Management : Manages the entire lifecycle of authentication tokens (saving, retrieving, decoding, removing).
• Secure API Requests : Works behind the scenes with the apiProxy to automatically attach authentication and CSRF tokens to every outgoing request.
• Centralized Configuration : Allows security policies like Content Security Policy (CSP) to be managed from a single configuration file.

With playguard , developers don't need to worry about the correct way to store a token or how to prevent cross-site scripting. They can simply use the provided methods and trust that the system is enforcing best practices.

Config Options

Global security policies are managed in config/play.security.config.json .

Config Key Table

Config Key	Default Value	Description	Recommended Value
csp	{...}	An object defining the Content Security Policy rules.	Keep as strict as possible
csrf.cookieName	"csrf_token"	The name of the cookie where the CSRF token is stored.	Match backend expectation
csrf.headerName	"X-CSRF-Token"	The name of the HTTP header used to send the CSRF token.	Match backend expectation
auth.storage	"cookie"	The storage mechanism for auth tokens.	"cookie"

Config Key	Default Value	Description	Recommended Value
auth.tokenName	"accessToken"	The name of the cookie or localStorage key used to store the token.	"accessToken"
auth.scheme	"Bearer"	The authentication scheme used in the Authorization header.	"Bearer"

Helper Methods

The helper provides a clean API for general security and authentication tasks.

Method Name	What It Does	Method Signature
sanitize	Cleans a string to make it safe for rendering as HTML text content. Strips HTML tags.	sanitize(input: string): string
purify	Strips dangerous HTML (like <script> tags and onclick attributes) from a string.	purify(html: string): string
auth.saveToken	Securely stores an authentication token.	auth.saveToken(token: string): void
auth.getToken	Retrieves the stored authentication token.	auth.getToken(): string
auth.removeToken	Deletes the authentication token, effectively logging the user out.	auth.removeToken(): void

Method Name	What It Does	Method Signature
auth.decodeToken	Decodes a JWT payload to read claims like user roles or expiration.	auth.decodeToken(): T
auth.isAuthenticated	Returns true if a valid, non-expired token exists.	auth.isAuthenticated(): boolean

Usage Examples

React: A Custom Authentication Hook

This useAuth hook encapsulates all authentication logic, providing a clean interface to components.

Angular: A Self-Healing Auth Interceptor

This HTTP interceptor automatically attaches the auth token to requests and handles 401 Unauthorized errors by logging the user out, creating a "self-healing" session management system.

Additional Info

Why We Created This Helper

Web application security is non-negotiable but notoriously difficult to get right. Without a dedicated helper, developers would be responsible for:

- Remembering to sanitize all user inputs to prevent XSS.
- Implementing secure storage for authentication tokens.
- Manually adding Authorization and CSRF headers to every API call.
- Writing boilerplate logic to handle token expiration and refresh cycles.

This is not only repetitive but also dangerously error-prone. playguard centralizes these critical security operations into a single, tested, and easy-to-use helper, ensuring that applications are secure by default, not by chance.

Best Practices & Developer Checklist

• Use apiProxy for all network requests. Check: Am I using apiProxy for all network requests instead of fetch() ?
• Sanitize all rendered user content. Check: Is all user-generated content that will be rendered as HTML passed through playguard.purify() ? For text content, am I using playguard.sanitize() ?
• Enforce a strict Content Security Policy (CSP). Check: Have I reviewed the csp rules in the config to ensure they are as strict as possible for my application?
• Use secure cookies for token storage. Check: Is my auth.storage strategy in the config set to "cookie" for production with the HttpOnly and Secure flags set on the server?
• Complement client-side security with backend validation. Check: Does our backend re-validate all data and permissions, treating the client as untrusted?

OWASP Top 10 Alignment

The playguard helper and associated patterns are designed to help mitigate several of the most critical web application security risks identified by OWASP.

OWASP Risk	How playguard Helps
A01: Broken Access Control	The apiProxy pattern ensures consistent attachment of auth tokens to every request.
A02: Cryptographic Failures	Promotes secure, HttpOnly cookie-based token storage over insecure localStorage .
A03: Injection (XSS)	The sanitize() and purify() methods are the primary defense against XSS.
A05: Security Misconfiguration	Centralizes security settings like CSP and CSRF headers in play.security.config.json .
A07: Identification Failures	The auth module standardizes the token lifecycle (save, get, remove, check).